LA-UR-79-1291

TITLE: COMMENTS ON THE CREATION OF SECURE OPERATING SYSTEMS

AUTHOR(S): J. C. Newell

STER

SUBMITTED TO: Association for Computing Machinery 7th Symposium on Operating Systems Principles

University of California

LOS ALAMOS SCIENTIFIC LABORATORY
Post Office Box 1663 Los Alamos, New Mexico 87545
An Affirmative Action/Equal Opportunity Employer

J. C. Newell
J-DCT
MS-668
X7 6475

Perspective

The intent of multilevel virtual operating systems is to provide each user with the logical equivalent of a distinct dedicated processing environment. Each user is connected to a process executing on a virtual processor within its own virtual address space.

I have chosen to be rather parochial in this treatment of secure operating systems with the intention of applying some of the wealth of excellent previous research to approach the development of a system tailored to the Department of Energy (DOE) and Department of Defense - type (DOD) large-scale scientific computing environments with which I am currently involved. This leads to the following assumptions which shape the underlying philosophy of this paper.

1) The computing environment often contains critical National Security Information (NSI) and thus, warrants substantial protection measures.

2) It is economically advantageous to share a single computer among users working at distinct and various levels of security privileges.

3) The necessary security policy is well-defined.

4) Personnel screening measures provide a reliably high level of staff integrity.

5) Reliable physical and data communication security is always present.

6) These systems must be considered to be constant targets of malicious penetration efforts.

7) It is desirable to make the cost of a security penetration effort as expensive as possible in terms of time and money.

8) The computer hardware to be employed is error-free with regard to predictable security-sensitive mechanisms.

This discussion does not attempt to present the technical details of this complex subject, but instead refers the reader to specific references.

The issue of information security arises when a single computer system provides computation and information storage to a community of users. As the cost and functional advantages of such virtual systems have been exploited, the need to protect the contained information has become extremely important. "Protection" is a general term describing all mechanisms which control access of a program to other objects internal to the system. There exists an enormous variety of such mechanisms. However, the computing community is well experienced with "secure operating systems" that have failed to measure up to their claims. "Security", in the context of operating systems, describes a set of negative attributes, e.g. making it impossible for a system user of a given security level to interfere with, or access the processes or information belonging to a user of greater security privilege. It is my contention that sufficient theory, knowledge and technology exist to produce an operating system that enforces the security policy required for the environment to which I have just alluded. I believe that the crux of this matter is the ability to verify the finished product and to prove the correctness of the security policy implementation.

Internal Protection Mechanisms

Security policy can be realized as a grouping of secured objects to which a given user should and should not be granted access under the distinct security levels. Secured objects are logical and physical elements of the computer system which are to be protected, e.g. files, processes, etc. Thus, in a rudimentary sense, a system can be declared secure if the required security policy can be described as sets of access relationships between device-type objects, data objects, and their users or as protected sets[1] and if the access protection is enforced. Internal protection controls are necessary to limit the object accessing capability of each virtual machine according to the individual users' authorization under the security policy.

Obviously, the internal protection mechanisms directly affect the security of the system. A multitude of such mechanisms have been devised, but historically their implementations have been within existing systems which were not designed with security in mind and the protection controls have been undermined by intrinsic logical flaws and design oversights inherent in the host system.[2] It is worthwhile to review some of the most thoroughly investigated mechanisms: controlled sharing and isolation, mutually suspicious subsystems, and access controllers and capability systems.

Perhaps the two most studied and implemented types of protection controls are isolation and controlled sharing. Generally, these controls attempt to rigorously define a physical[3] or spatial[4] domain in which a particular process executes at any given instant. This domain is most often achieved through sets of access capabilities coupled with hardware mechanisms, e.g. Multics-like descriptor registers.[5] Isolation restricts a process to access only the objects within its assigned domain. Virtual machine techniques are well suited to supporting isolation mechanisms[6] and are well understood. Whereas isolation attempts to maintain a logical "fire wall" between co-existing virtual machines, controlled sharing implies permitting a process executing in one domain to have restricted access capability to objects in another domain. Controlled sharing constitutes a more complex implementation problem than does isolation. Careful consideration of the benefits and costs in terms of design complexity, protection reliability, and system performance will likely indicate that controlled sharing is not an absolute necessity in the specific environment that I am addressing.

Mutually suspicious subsystems (or "protected subsystems") are characterized by a requirement for a process to be allowed to switch from one execution domain to another while retaining only a subset of access capabilities from its prior domain. The objective in this situation is to provide mutual protection for multiple subprocesses (i.e. multiple domains) that must cooperate within the context of a single virtual machine. This attribute facilitates the secure use of borrowed programs among users and foreign programs (i.e. products of external organizations not under the control of the host system's security staff) by supporting encapsulation and confinement constraints on the subject processes. The work of Jones,[7] Lampson,[8] and Schroeder[9] lends credence to the viability of implementing multiple synchronous cooperating subprocess domains within a single virtual machine.

Access controllers[10] and capability-based[11,9] addressing schemes, either individually or in combination, have been explored in many different applications and implementations. Access controller systems are list-oriented and typically containe , at the least, an address descriptor for the storage segment containing the list and the list itself. When a segment of storage is to be accessed by a user, a separate "shadow" structure can be established simultaneously to contain the access controller mechanism. The controller's address descriptor is assigned a unique designator or identifier that is inserted into the memory mapping system that directs the location of secured objects. Then in order for the processor to accomplish a users' request to access a secured object, the unique designator of the appropriate access controller must be passed from the processor to the memory system and individual accesses checked for consistency with the access control list. Capability systems also have numerous permutations, but all are basically ticket-oriented mechanisms. A useful implementation would load a protection descriptor register for a given virtual process with the unique capability, i.e. a "ticket" that creates a secure path to the catalog of capabilities which controls the specific segments that the process is entitled to access. Each capability in a catalog should include designators for the specific access privileges granted to the user for each segment. The research of Saltzer[12] and Fabry[11] shows that the most effective implementations of capability systems is through special hardware, such as a tagged storage architecture, that can prevent forging or illicit manipulation of capability values. Access controller and capability systems have many design and implementation subtitles arising from hardware architecture and attempts to provide authorization to share segments which implies changing of lists or capabilities. Capability controls in particular show significant promise and methods to cope with these subtle difficulties have been devised.[13] A carefully formulated straightforward security policy doe much to ensure that these protection mechanisms can be implemented efficiently and with reliable integrity.

Major Threats to a Military Security System

Historically, and in the context of an adversary philosophy, the two most insidious subversion techniques probably have been so-called "Trojan horses" and "trapdoors". The term Trojan horse refers to a covert software feature which is designed to allow a penetrator to circumvent the internal protection mechanisms of the system without actually needing access to the system's internals. An example of a simple Trojan horse is a program which emulates the interactive user sign-on and authentication procedure of a time-sharing system. If the emulator is left in control of a terminal which will be used by a person with specially classified passwords or identifiers, then those classified items may be captured by the emulator which in turn exits and gives the user a realistic error message so that he is unaware of the subversion. This case is relatively straightforward and can be protected against. However, Trojan horses can be added after a system implementation is complete and installed. One possibility is the camouflaging of the covert feature within a mathematical library subroutine such that when invoked, it may access privileged information or data. A Trojan horse may also be used to implant a trapdoor in an operating system long after the system has been installed and when it is less likely to be discovered.

"Trapdoor" describes covert code, which is embedded into an operating system, usually requiring high level access to operating system design details, that facilitates bypassing the protection mechanisms from a far removed level, e.g. unclassified user level. Trapdoors may be inserted during design or indirectly after implementation and testing by modifying the system source code so that the code is automatically included during a subsequent recompilation of the source. Guarding against trapdoors is a complex and costly undertaking.

It appears that the range of deliberate system attack scenarios that can be instigated by adversaries are considerably fewer than the range of possibilities for unintentional flaws that may occur while designing or implementing a system as complex as a large scale multilevel multiprogramming operating system. The operational behavior of the integrated hardware subsystems may also manifest idiosyncracies that can be exploited by adversaries. A prime source of unintentional logical flaws within complex software systems are dependency structures. Shroeder, Clark, and Saltzer[14] classify dependencies as explicit (i.e. caused by procedure calls or interprocess communications) and implicit (i.e. caused by direct physical sharing of writeable data among managers of secured objects). If the correct functioning of a code module which manages secured objects depends upon the prior correct functioning of another module, then a dependency exists. Schroeder, et. al. have delineated five categories of dependencies that should be considered when quantifying the total dependency structure of a set of integrated "object manager" modules. These categories are:

1) Component Dependencies: A module X depends upon other modules that control secured objects that are components of the objects defined by X.
2) Map Dependencies: A module X depends upon other modules that provide the map of correlations between the names of the objects X controls and the names of the components of each object.
3) Program Storage Dependencies: A module X contains algorithms which use information stored as secured objects that are controlled by other modules.

4) Address Space Dependencies: A module depends upon other modules which control the address space (i.e. object) in which the module executes.

5) Interpreter Dependencies: A module depends upon another module that implements the interpreter or virtual processor required for the module to execute.

The majority of protection mechanism flaws which have been encountered during attempts to penetrate existing conventional operating systems with security controls that were added after design and implementation of the general purpose system was completed can be attributed to these general classes of dependencies. The experiences of Attanasio, et al,[15] Popek and Farber,[1] and Shroeder, et al[14] are relevant to the development of a secure system. McPhee[16] suggests some solutions to dependencies found in the first release of OS/VS2 by IBM and in particular the Time-of-Check-to-Time-of-Use (TOCTTOU) problem. The TOCTTOU problem arises in multiprogramming systems in which a time interval exists between a validity check of specific secured objects and the completion of the operation to be performed with those objects. An example might involve timing idiosyncracies of an input/output (I/O) system segment and its associated I/O channel protocol or command list processing that might allow validity-checked objects to be altered before the I/O operation process was completed. Virtual memory paging mechanisms can also introduce flaws if not properly understood and controlled.

Traditional Approaches to Achieving Computer System Security

The realization that existing multiprogramming time-shared systems in the late 1960's afforded little in terms of system security and were riddled with protection flaws and weaknesses spurred substantial research efforts into physical and logical (i.e. internal) computer operating system security that continue to the present. Unfortunately, in those early years the science of designing and implementing large scale complex software systems was not in the state of sophistication, advancement, and acceptance that it is today. This fact coupled with an early emphasis on physical protection rather than the emerging security kernel concepts, modular design, virtual machine theory, verifiability, etc. no doubt contributed to a number of disappointing experiences (as perceived by users, e.g. the Military) with computer security research. A good deal of skepticism still persists in security-conscious organizations such as the Government and Military and not enough support has been given to applying recent developments in theory, technology, and software engineering in a coherent manner to create modern solutions that can enforce security policies. Physical protection methods are very well developed and are exemplified by communications cryptographic technology and computer facility operation procedures for secure environments. Although external protection techniques are known, it is not clear they are employed widely or uniformly so as to be thoroughly effective.

In order to deal expediently with obvious security and integrity shortcomings in operating systems used in the NSI community, three popular solutions were proposed in the forms of restricted operating procedures, audit and surveillance subsystems, and tiger teams. These approaches, however, do not produce secure operating systems with high degrees of integrity. Restriction of operating procedures is typified by the operation of military multiuser systems at one exclusive information security level. Before the system can be used for processing at a different level it must be purged of all information not authorized for the new security level. This procedure is time consuming and typically involves a great deal of manual intervention to exchange removable storage media, etc. In many cases this single-level operation coupled with the purging of tasks leads to inefficient and extremely costly computer operations. In 1973 the expense caused by restricted operating procedures within the Air Force alone was estimated to be $100,000,000 per year.[2]

Audit and surveillance subsystems are to be included in an operating system to detect and report breaches in security policy within the system. Early attempts to implement such subsystems were ineffective due to the inability to precisely formulate and model system security policy so that the surveillance subsystem could be programmed to recognize illicit actions when they occurred. This severely limited the scope of surveillance possible. Further, the absence of fully developed security specification techniques and logic verification methods meant that the subsystem itself could not be relied upon to resist subversion completely. More recent work by Jones and Lipton[17] describes audit/surveillance subsystems in the proper perspective and indicates mechanisms that may make these subsystems effective and desirable. One serious consideration of these subsystems is their effect on system performance and validity checking overhead.

Tiger teams are groups of skilled systems architects and programmers who attempt to penetrate a functioning operating system in order to discover flaws in the internal protection mechanisms. Tiger teams do have a viable mission, but it must be recognized that they can only prove the presence of flaws and not their absence. When flaws are discovered they are to be alleviated, but the act of correcting one flaw may simultaneously introduce undesirable unknown side effects. This means that tiger team assaults must be an exhaustive iterative process that is not generally a cost effective or productive method of creating a secure system.

An Approach to the Overall Problem of Creating Secure Operating Systems

It can be argued that the rapid advances in computer hardware technology and steady decline in price/performance ratios reduce the need for secure multilevel multiuser systems since each level of usage can be given substantial computational resources at relatively low cost with individual isolated systems and restricted operating procedures. I will assume that the large-scale scientific computing environment that is concerned with NSI will not significantly benefit from this economic trend for several years to come and that the development of secure multilevel virtual systems will produce results that can also be applied to further reducing the cost and management burden of Federal and commercial systems in the future.

Due to the marked advances in recent years in the fields of data processing system protection and security, system structure and correctness, software engineering and management, and software reliability and correctness; I submit that secure operating systems with high degrees of integrity for the NSI environment are now within our grasp. By "security", I refer to the ability of an operating system to embody a specific information protection policy and control the system resources to prevent the accessing or modifying of protected information by adversary (i.e. malicious) users. "Integrity" is the quality of soundness or correct implementation of a specific security policy. Designing and implementing such a system should not be construed as a trivial or inexpensive process, but it is feasible. The following discussion takes a broad overview of a NSI computing environment (see opening remarks) starting at the physical perimeter and working inward toward the central computer operating system while enumerating the principal design areas where existing theory can be directly applied.

The hypothetical environment will include a central facility under strict physical security measures, i.e. guards, fences, etc. All personnel at this facility will also have been carefully screened and cleared by a stringent program, e.g. DOE Q-clearance. Assume that all of these personnel are authorized to the highest security level, but that need-to-know rules and special subclassifications control dissemination of classified information. In this facility it is reasonable to assume that an adversary able to gain clearance would have other means to access sensitive information other than through a computer system, e.g. reports, waste containers, desks, etc. Nevertheless, in this environment the terminal access can be reliably limited to persons already screened by the clearance process and admitted through physical security. The most crucial requirement here is to enforce the need-to-know and subclassification criteria of the installation security policy to prevent, for example, users authorized for Unclassified (U) or Secret (S) access from accessing or modifying Top Secret (TS) information. A second facility adjunct to the central facility which must be operated as an open (i.e. totally unclassified) environment, e.g. an international research laboratory. The presence of cleared as well as uncleared personnel, all of whom are to be restricted to U information and computing resources, can present a convenient path for access by an adversary. One solution might be to provide this facility with a distinct isolated computer system physically separated from the central facility. It will be presumed that this is not desirable for economic, space, and very large-scale computational capability requirements so that the vast resources of the central facility are necessary.

The principal objective of this overall system is the implementation of the total security policy within the physical constraints of single computer system housed within the central facility and used to process TS, S, and U information. Computer system access is to be through data terminals as in a conventional interactive time-shared system. Beginning with the outermost perimeter of the system, we first encounter the terminal subsystem. In many ways this is the best understood and most straightforward system to protect. Data communication lines can be physically shielded and protected and the channel information should be encrypted. At this level conventional encryption techniques are probably more than adequate.[20] The lines should also be attached to specific fixed locations, e.g. offices and other designated areas to aid in authenticating the user's environment and privileges. Signature verification based upon the unique unforgeable acceleration and rhythm cues of the user's handwriting holds future promise for authenticating the identity and privileges of each user when he presents his assigned unique identifiers to the system. The focal point of the terminal subsystem should be a dedicated device which implements the user location/identifier/access authorization authentication portion of the security policy. All communication lines entering the system must pass through this filter first. At this point access to the operating system can be denied and data terminal surveillance mechanisms are implemented. If the filter system is designed and implemented for certification in the manner to be proposed later for the central operating system itself, according to a precisely defined and modeled security policy, then it can be made secure. The results can be predicted to a certain extent, by examining the success of the Network Security Controller developed by the Los Alamos Scientific Laboratory and the MITRE Secure Communications Processor project.[18,19] The controlled terminal interface subsystem routes specific authenticated user communications to separate distinct line interface hardware and software associated with each classification environment within the central computing system.

With the transmit and receive data communication paths established, attention can be focused on the crucial issues concerning the central computing operating system and its machine environment. It is imperative that before the system specification and design is begun the security policy must be precisely defined and translated into a form which facilitates verification and implementation. The classic models of Bell and LaPadula[21,22] are not complete for the typical NSI computing environment, are unnecessarily complex, and do not lend themselves easily to automated manipulation and verification. One of the more viable and easily managed modeling methodologies that is also suited to automated manipulation and proof is that proposed by Feirtag, et al[23] and expressed in the language SPECIAL.[24] Other recent axiomatic approaches such as the simple and flexible technique suggested by Jones and Lipton[17] are also candidates. The important issue in this preliminary phase of system development is that the security policy must be defined and described clearly, precisely, and concisely so that it is verifiable and intellectually manageable. If these qualities are not achieved there is little hope for implementing a "secure" operating system.

The development group itself should be structured in a manner that can protect the evolving system from potential threats such as trapdoor implacement. The project staff should be cleared to the highest level at which the final system may be used. The two-man rule (or "buddy system") should be employed throughout development and use. This approach requires the cooperation of two persons to access or modify an object. Double-encryption is one method of implementing the two-man rule. Public-key cryptosystems based upon the difficulty of factoring large numbers as advocated by Rivest, et al[25] may also be applied depending upon the development organization and environment. All design and development work should be performed on a computer system under restricted operating procedures, dedicated to the project and controlled solely by the project administration. The system software staff should be kept as small as possible and organized as teams with specific module responsibility and accountability. One "security team" of perhaps four senior systems staff (including the chief system programmer) should be dedicated to verification of all modules and enforcement of the security policy within the operating system. This team functions as two closely interacting groups to implement a "check and balance" arrangement during individual module development. When a module is tested and declared to be properly functioning by its implementer, it is turned over to the security team. Each two-person subgroup of this team possesses a different classified conventional encryption key. After verifying the module against the security model, the two subgroups doubly-encrypt the module along with its associated verification analysis and checksum and place it in restricted storage. Any further manipulation of that module requires the mutual cooperation of both subgroups or undermining of both crytographic keys. This process is repeated during system testing until final system integration and installation is appropriate. At this time the security team functions as a unit in final verification and certification of the system and then the chief staff member encrypts it with a single key after which the chief security officer for the target system also encrypts the system with a different key. Loading and initialization of the system then requires two-man cooperation. At this point the system can be placed into operation and subjected to tiger teams and surveillance.

Certain important design characteristics of the operating system are expressed in the following general design philosophy:

a) The system must be logically verifiable with automatic or semi-automatic tools[1] to demonstrate the security policy is correctly implemented and enforced. Such a certification should be compatible with a lattice model as suggested by Denning[26] and Denning.[27]

b) The system should be structured hierarchically[6] to segregate user applications, user systems, the system control program, and the security mechanism. Davies' concept of "spheres of control"[28] i.e. the logical boundaries and properties associated with each active system element, can be applied to defining and formalizing the hierarchy.

c) The security policy should be implemented exclusively as a protected nucleus, compact ar verifiable, whose integrity guarantees the enforcement of the policy and which acts as the hub of the systems structural organization. This kernel concept has been substantiated by many including Schroeder,[29] Popek and Farber,[1] Popek and Kline,[30] and Ames[31] and permits development of the upper levels of the hierarchy without having to reevaluate the security of the entire system.

d) The operating system and especially its security kernel must be intellectually manageable, as with any sophisticated system. Tools and languages that facilitate this discipline are the subject of intense study and experimentation and should be applied to secure systems whenever appropriate. Guttag, et al[32] have developed an algebraic axiomatic structure that serves to reduce the amount of complexity that must be comprehended for any process in a given context. Such abstraction reinforces the security kernel verification process and improves the feasibility of the task.

e) The inherent clarity, implicity, and maintainability of the virtual machine concept for a multiuser monoprocessor system[30] makes it the logical basis for this system. In the event that a multiprocessor system becomes necessary to replace individual virtual processors, Fabry's research suggests some techniques which might aid the transformation without degrading reliability or integrity.

The detailed design phase of a secure operating system must consider the many known difficulties and attempt to develop solutions in light of past research and experience. These technical considerations are too numerous to detail in this discussion, but for the hypothetical system being described some particularly challenging areas can be identified.

a) The functional interrelationships between the security kernel and its upper hierarchical layers [e.g. the virtual machine monitor (VMM)] must be carefully studied. Wulf, et al[33] points out the subtle difference between system policies and the mechanisms used to realize them. An important example of such a subtlety involves the handling of user I/O. Although the security kernel must be directly involved with I/O transfers as they affect security, the I/O processes and device handlers need not be integrated into the kernel.[34] Such processes as I/O, scheduling, allocation, etc. are best implemented in the VMM. The VMM can be kept modular and manageable without lessening security kernel integrity.

b) I/O subsystem integrity can be enhanced by segregating I/O devices such as disc and tape systems into separate TS, S, and U compartments. Run-time validity checking can also be performed within the I/O process at the cost of additional complexity to ensure the security of individual virtual user processes that are executing in the same virtual processor.

c) Close attention must be given to eliminating loops in dependency structures, i.e. the structure of the dependent relationship should be a partially ordered set.[14]

d) Capability-based addressing[11] has the potential of offering a highly efficient form of absolute address for virtual secured objects and simplified programming conventions. The implementation should be considered in the light of the dynamic capability-list management problems encountered by Lampson and Sturgis in the Cal system.[35]

e) All operating system structuring, including the security kernel, must be viewed from the standpoint of yielding high operating efficiency and correct use of the hardware features.

It is most probable that little or no freedom exists in procuring a new computer hardware system or designing a totally new system specifically to implement the hypothetical secure operating system, but rather it will have to be implemented with an existing system. However, one can identify desirable hardware features that can be useful in implementing a virtual secure system. No single system currently has all the desirable features, but some are much better suited to security objectives, e.g. the IBM System/370, Honeywell Level 68, and DEC KL-10. A few desirable architectural hardware features are listed below.

a) Virtual memory, either page or segment oriented
b) High-speed dynamic address translation
c) Hardware-enforced processor states
d) Tagged-memory architecture
e) Mulitple register sets
f) High-speed CPU control registers
g) Nonrepetitive time-of-day clock
h) Error-trapping and reporting
i) Programmable I/O channel control

The Virtual Control Storage (VCS) feature is an experimental extension to the IBM VM/370 operating system implemented in software. Attanasio's discussion of VCS[36] includes hardware analogies to programmable control storage concepts. A VCS-type facility, implemented in hardware control memory could utilized to provide a special protected high-speed context or domain switching capability and to implement various virtual processor functions as execute-only synchronous hardware instructions. The VCS concept has many novel implications.

Conclusion

I have reviewed the predominant general classes of problems that impact secure operating systems for NSI or Military-type environments and cited research results that can be directed toward solving those problems in a multiuser multilevel-security virtual system. Although many technical issues remain to be resolved and the production of a secure operating system is still expensive, the state of the art of system design makes such a project feasible and puts a class of verifiably secure virtual systems within our grasp. My intention for these comments is to renew awareness and interest in secure systems within agencies and organizations which may have given up hope for secure systems or have not considered the potential and to generate discussion of a general systematic approach to creating an NSI-level secure system.

Bibliography

[1] Gerald J. Popek and David A. Farber, "A Model for Verification of Data Security in Operating Systems," Commun. ACM, Vol. 21, no.9, pp. 737-749, September 1978.

[2] J. P. Anderson, "Computer Security Technology Planning Study," ESD-TR-73-51, Vols. and II, October 1973.

[3] R. Fabry, "Dynamic Verification of Operating System Decisions," Commun. ACM, Vol. 16, no.11, pp. 659-668, November 1973.

[4] P. J. Denning, "Third Generation Computer Systems," Computing Surveys, Vol. 3, no.4, pp. 175-216, 1971.

[5] Elliot Organick, "The Multics System: An Examination of its Structure," M.I.T. Press, 1971.

[6] J. J. Donovan and S. E. Madnick, "Hierarchical Approach to Computer System Integrity," IBM Systems Journal, Vol. 14, no.2, pp. 188-202, 1975.

[7] A. Jones, "Protection in Programmed Systems," Ph.D. dissertation, Carnegie-Mellon Univ., 1973.

[8] B. W. Lampson, "Dynamic Protection Structures," 1969 FJCC AFIPS Conf. Proc., Vol. 35, pp. 27-38

[9] M. D. Schroeder, "Cooperation of Mutually Suspicious Subsystems in a Computer Utility," Project MAC TR-104, M.I.T.,1972.

[10] A. Bensoussan, C. Clingen, and R. Daley, "The Multics Virtual Memory: Concepts and Design," Commun. of ACM, Vol.15, no.5, pp.308-318, May 1972.

[11] R. S. Fabry, "Capability-based Addressing," Commun. of ACM, Vol.17, no.7, pp. 403-412, July 1974.

[12] J. H. Saltzer, "Traffic Control in a Multiplexed Computer System," Project MAC TR-30, M.I.T., 1966.

[13] J. H. Saltzer and M. D. Schroeder, "The Protection of Information in Computer Systems," Proc. of the IEEE, Vol.63, no.9, pp. 1278-1308, September 1975.

[14] M. D. Schroeder, D. D. Clark, J. H. Saltzer, "The Multics Kernel Design Project," Proc. of the 6th ACM Symposium on Operating Systems Principles, Vol.11, no.5, pp.43-56, November 1977.

[15] C. R. Attanasio, P. W. Markstein, and R. J. Phillips, "Penetrating an Operating System: A Study of VM/370 Integrity," IBM Systems Journal, no.1, pp. 102-116, 1976.

Continuation of Bibliography

[16] W. S. McPhee, "Operating System Integrity in OS/VS2," IBM Systems Journal, no.3, pp. 230-252, 1974.

[17] A. K. Jones and R. J. Lipton, "The Enforcement of Security Policies for Computation," Proc. of ACM 5th Symposium on Operating Systems Principles, Vol.9, no.5, pp.197-206.

[18] P. S. Tasker, "Design of a Secure Communications Processor: Overall Environment and Concept," Air Force Systems Command Electronic Systems Division, ESD-TR-73-135, Vol. 1, May 1973.

[19] P. S. Tasker, "Design of a Secure Communications Processor: Central Processor," Air Force Systems Command Electronic Systems Division, ESD-TR-74-181, Vol. 3, June 1974.

[20] R. M. Needham and M. D. Schroeder, "Using Encryption for Authentication in Large Networks of Computers," Commun. of ACM, Vol. 21, no. 12, pp. 993-999, December 1978.

[21] D. E. Bell and L. J. La Padula, "Secure Operating Systems: Mathematical Foundations," Air Force Systems Command Electronics Systems Division, ESD-TR-73-278, Vols. I and II, November 1973.

[22] D. E. Bell, "Secure Computer Systems: A Refinement of the Mathematical Model," Air Force Systems Command Electronic Systems Command, ESD-TR-73-278, Vol. III, April 1974.

[23] R. J. Feirtag, K. N. Levitt, and L. Robinson, "Proving Multilevel Security of a System Design," Proc. of the Sixth ACM Symposium on Operating Systems Principles, Vol. 11, no. 5, pp. 57-65, November 1975.

[24] O. Roubine and L. Robinson, "SPECIAL Reference Manual," Stanford Research Institute, 1977.

[25] R. L. Rivest, A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public-key Cryptosystems," Commun. of ACM, Vol. 12, no. 2, pp. 120-126, February 1978.

[26] D. E. Denning, "A Lattice Model of Secure Information Flow," Commun. of ACM, Vol. 19, no. 5, pp. 236-242, May 1976.

[27] D. E. Denning and P. J. Denning, "Certification of Programs for Secure Information Flow," Commun. of ACM, Vol. 20, no. 7, pp. 504-513, July 1977.

[28] C. T. Davies, "Data Processing Spheres of Control," IBM Systems Journal, Vol. 17, no. 2, pp. 179-198, 1978.

[29] M. D. Schroeder, "Engineering a Security Kernel for Multics," Proc. of the Fifth ACM Symposium on Operating Systems Principles, Vol. 9, no. 5, pp. 25-32, November 1975.

Continuation of Bibliography

[30] G. J. Popek and C. S. Kline, "Verifiable Secure Operating System Software," AFIPS, Proc. of the 1974 National Computer Conference, pp. 145-151, 1974.

[31] S. R. Ames, "The Design of a Security Kernel," The MITRE Corp., M75-212, April 1975.

[32] J. V. Guttag, E. Horowitz, and D. R. Musser, "Abstract Data Types and Software Validation," Commun. of ACM, Vol. 21, no. 12, pp. 1048-1064, December 1978.

[33] W, Wulf, E. Cohen, W. Corwin, A. Jones, R. Levin, C. Pierson, and F. Pollack, "HYDRA: The Kernel of a Multiprocessor Operating System," Commun. of ACM, Vol.17, no. 6, pp. 337-345, June 1974.

[34] E. L. Burke, "Concept of Operation for Handling I/O in a Secure Computer at the Air Force Data Services Center," Air Force Systems Command Electronic Systems Division, ESD-TR-74-113, April 1974.

[35] B. W. Lampson and H. E. Sturgis, "Reflections on Operating System Design," Commun. of ACM, Vol. 19, no. 5, pp. 251-265, May 1976.

[36] C. R. Attanasio, "Virtual Control Storage - Security Measures in VM/370," IBM Systems Journal, Vol. 18, no. 1, pp. 93-110, Jnauary 1979.